(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 817 081 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
07.01.1998 Bulletin 1998/02

(51) Int. Cl.[6]: **G06F 12/08**

(21) Application number: **97304796.2**

(22) Date of filing: **30.06.1997**

(84) Designated Contracting States:
AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE

(30) Priority: **01.07.1996 US 673881**

(71) Applicant:
SUN MICROSYSTEMS, INC.
Mountain View, California 94043-1100 (US)

(72) Inventors:
- **Hagersten, Erik E.**
  Palo Alto, California 94303 (US)
- **Guzovskiy, Aleksandr**
  Lowell, Massachusetts 01852 (US)

(74) Representative:
Turner, James Arthur et al
D. Young & Co.,
21 New Fetter Lane
London EC4A 1DA (GB)

(54) Flushing of cache memory in a computer system

(57) An efficient streamlined cache coherent protocol for replacing data is provided in a multiprocessor distributed-memory computer system. In one implementation, the computer system includes a plurality of subsystems, each subsystem includes at least one processor and an associated cache and directory. The subsystems are coupled to a global interconnect via global interfaces. In one embodiment, when data is replaced from a requesting subsystem, an asynchronous flush operation is initiated. In this implementation, the flush operation includes a pair of decoupled local flush instruction and corresponding global flush instruction. By decoupling the local flush instructions from the global flush instructions, once the requesting processor in the requesting subsystem is done issuing the local flush instruction, the requesting processor does not have to wait for a corresponding response from home location associated with the data being replaced. Instead, the requesting processor is freed up quickly since there is no need to wait for an acknowledgment from the home location (home subsystem) over the global interconnect. The home subsystem responds with an appropriate ACK message. The requesting subsystem reissues a read-to-own (RTO) transaction on its local interconnect thereby retrieving and invalidating any copy(s) of the data in the requesting subsystem. A Completion message is sent to the home subsystem together with the dirty data. Subsequently, a confirmation of the completion of the flush operation can be implemented using a "synchronization" mechanism to verify that all previously valid cache lines associated with a page have been successfully replaced with respect to their home location and the replaced cache lines are now marked "invalid" at the home subsystem.

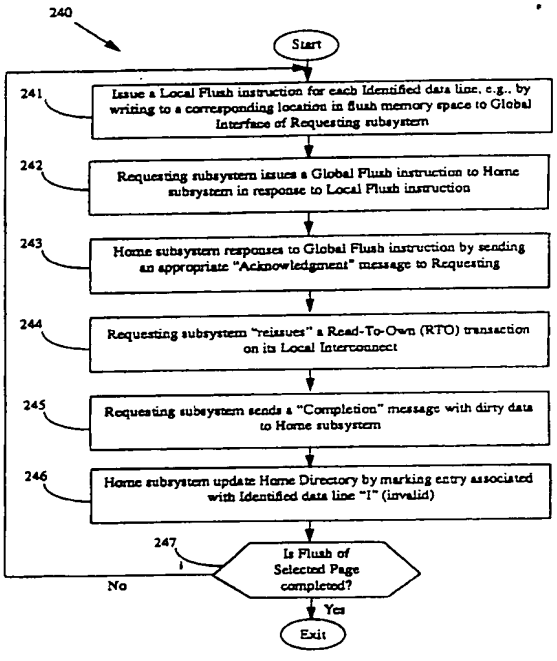

FIG. 2B

EP 0 817 081 A2

Description

This invention relates to caches in computer systems.

In a multi-level cache computer system having at least a lower-level cache and a higher-level cache, since the cache sizes are not infinitely large, eventually it becomes necessary to replace duplicated data in the computer system's cache memory in order to make room for caching new data. Generally, the smaller lower-level cache can replace data in its cache lines by generating write-backs, while replacement of the cached data pages in the larger higher-level cache is done under software control.

In one simplistic scheme, when a page in the higher-level cache memory needs to be replaced from a requesting subsystem of the computer system, the following sequence of steps are performed. For every cache line associated with the page, regardless of the status of the cache line, a "replace_request" message is propagated, all the way to the data's home location. The home location references a home directory to determine the status of the cache line. If the requesting subsystem has "dirty" data, a request for the data is made from the home location to the requesting subsystem. The requesting subsystem then provides the data to the home location. Upon receipt of the data, the home location marks the appropriate entry of the home directory "Invalid" and a "replace_completed" message is sent back to the requesting subsystem.

Unfortunately, the above-described simplistic scheme generates an excessive amount of network traffic because an unnecessary number of network messages are exchanged between the requesting subsystem and the home location.

Thus there is a need for an efficient mechanism for replacing data in the cache memory of a computer system which maintains data coherency while reducing network traffic within the computer system.

Various respective aspects and features of the invention are defined in the appended claims.

Embodiments of the invention relate to a mechanism for maintaining data coherency when replacing data in the caches of computer systems.

Embodiments of the present invention provide an efficient streamlined cache coherent protocol for replacing data in a multiprocessor distributed-memory computer system. In one implementation, the computer system includes a plurality of subsystems, each subsystem includes at least one processor and an associated cache and directory. The subsystems are coupled to a global interconnect via global interfaces.

In one embodiment, when data is replaced from a requesting subsystem, an asynchronous flush operation is initiated. In this implementation, the flush operation includes a pair of decoupled local flush instruction and corresponding global flush instruction. By decoupling the local flush instructions from the global flush instructions, once the requesting processor in the requesting subsystem is done issuing the local flush instruction, the requesting processor does not have to wait for a corresponding response from home location associated with the data being replaced. Instead, the requesting processor is freed up quickly since there is no need to wait for an acknowledgment from the home location (home subsystem) over the global interconnect.

In this embodiment, the home subsystem responds with an appropriate ACK message. The requesting subsystem reissues a read-to-own (RTO) transaction on its local interconnect thereby retrieving and invalidating any copy(s) of the data in the requesting subsystem. A Completion message is sent to the home subsystem together with the dirty data.

Subsequently, a confirmation of the completion of the flush operation can be implemented using a "synchronization" mechanism provided by the computer system. Once such confirmation verifies that all previously valid cache lines associated with a page have been successfully replaced with respect to their home location, then the replaced cache lines can now be marked "invalid" at the home subsystem.

The invention will now be described by way of example with reference to the accompanying drawings, throughout which like parts are referred to by like references, and in which:

Figure 1A is a block diagram showing a networked computering system 100 with a hybrid cache-only memory architecture/non-uniform memory architecture (COMA/NUMA).

Figure 1B is an exemplary memory map for a networked computering system of Figure 1A.

Figures 2A and 2B are flowcharts illustrating one embodiment of the invention.

Figure 3 is a protocol table depicting the operation of the embodiment illustrated by Figure 2.

Figure 4 is a block diagram depicting one embodiment of the global interface of Figure 1A.

Figure 5 is a block diagram of one embodiment of one portion of the global interface of Figure 4.

Figure 6 is a table depicting synchronous operations employed by one embodiment of the computer system of Figure 1A.

Figure 7 is an exemplary code sequence using one of the synchronization operations shown in Figure 6.

In the following description, numerous details provide a thorough understanding of the invention. These details include functional blocks and an exemplary cache architecture to aid implementation of a cost-effective scheme for maintaining data coherency within a computer system. In addition, while the present invention is described with reference to a specific data coherent scheme for a distributed cache of a multiprocessor

computer system, the invention is applicable to a wide range of caches and network architectures. In other instances, well-known circuits and structures are not described in detail so as not to obscure the invention unnecessarily.

A hybrid cache-only memory architecture/non-uniform memory architecture (COMA/NUMA) having a shared global memory address space and a coherent caching system for a networked computing system is illustrated in Figure 1A showing one such hybrid COMA/NUMA computer system 100 which provides a suitable exemplary hardware environment for the present invention.

System 100 includes a plurality of subsystems 110, 120, ... 180, coupled to each other via a global interconnect 190. Each subsystem is assigned a unique network node address. Each subsystem includes one or more processors, a corresponding number of memory management units (MMUs) and hybrid second level caches (L2$s), a COMA cache memory assigned with portion of a global memory address space, an optional third-level cache (L3$), a global interface and a local interconnect. For example, subsystem 110 includes processors 111a, 111b ... 111i, MMUs 112a, 112b, ... 112i, L2$s 113a, 113b, ... 113i, COMA cache memory 114, L3$ 118, global interface 115 and local interconnect 119. In order to support a directory-based cache coherency scheme, subsystems 110, 120, ... 180 also include directories 116, 126, ... 186 coupled to global interfaces 115, 125, ... 185, respectively.

Data originating from, i.e., whose "home" location is, anyone of COMA cache memories 114, 124, ... 184 may be duplicated in cache memory of system 100. For example, in COMA mode, system 100's cache memory includes both COMA cache memories 114, 124, ... 184 and L2$s 113a ... 113i, 123a ... 123i, and 183a ... 183i, and data whose "home" is in cache memory 114 of subsystem 110 may be duplicated in one or more of cache memories 124, ... 184 and may also be duplicated in one or more of L2$s 113a ... 113i, 123a ... 123i, and 183a ... 183i. Alternatively, in NUMA mode, system 100's cache memory includes L2$s 113a ... 113i, 123a ... 123i, and 183a ... 183i, and data whose "home" is in cache memory 114 of subsystem 110 may be duplicated in one or more of L2$s 113a ... 113i, 123a ... 123i, and 183a ... 183i.

In one embodiment of the present invention, as illustrated by the hybrid COMA/NUMA computer system 100 of Figure 1A and the memory map of Figure 1B, the "home" location of a page of data is in COMA cache memory 124 of subsystem 120, i.e., subsystem 120 is the home subsystem. The content of the home page can also exist in the cache memory space of one or more of requesting subsystems, for example, in the memory space of requesting subsystem 110. Hence, in COMA mode, memory space is allocated in COMA cache memory 114 in page increments, also known as shadow pages, and optionally in hybrid L2$s 113a, 113b, ... 113i in cache line increments. Alternatively, in NUMA mode, memory space can be allocated in hybrid L2$ 113a, 113b, ... 113i in cache line increments. Note that the allocation of memory units in system 100, pages and cache lines, are only exemplary and other memory units and sub-units are possible.

Home directory 126 is responsible for maintaining the states of existing copies of the home page throughout system 100. In addition, MTAGs associated with the home memory and any shadow page in subsystems 110 and 180, track the status of the local copies in each requesting subsystem using one of the following four exemplary states.

An **invalid** ("I") state indicates that a particular subsystem does not have a (cached) copy of a data line of interest.

A **shared** ("S") state indicates that the subsystem, and possibly other nodes, have a shared (cached) copy of the data line of interest.

An **owned** ("O") state indicates that the subsystem, and possibly other nodes, have a (cached) copy of the data line of interest. The subsystem with the O copy is required to perform a write-back upon replacement.

A **modified** ("M") state indicates that the subsystem has the only (cached) copy of the data line of interest, i.e., the subsystem is the sole owner of the data line and there are no S copies in the other nodes.

Figures 2A and 2B are flowcharts and Figure 3 is a protocol table, illustrating how global data coherency between a shadow page in cache memory 114 and its corresponding home page in cache memory 124 is maintained, when requesting subsystem 110 needs to free the memory space currently occupied by the shadow page. Note that while the following example describes a flush operation on cache lines associated with a page and cached in a higher-level cache, e.g., cache memory 114, the invention is applicable to other computer systems with non-COMA/NUMA architectures, such as a computer system with a COMA-only or any other type higher-level cache.

The following column definitions provides a guide to using the protocol table of Figure 3.

Bus Trans specifies the transaction generated on the local interconnect. A *Writestream* to alternate LPA space have extra mnemonics added: prefetch shared (WS_PS), prefetch modified (WS_PM), fast write (WS_FW) and flush (WS_FLU).

Req. Node MTAG tells the MTAG state of the requested cache line, e.g. M ($M_{ODIFIED}$). O (O_{WNED}), S (S_{HARED}) or I (I_{NVALID}). Accesses to remote memory in NUMA mode have no valid MTAG and are denoted N (NUMA) in this column.

Request specifies what transactions are sent from

the requester to the home agent.

State in Dir describes the *D-state*, which is the state in which the requesting node is (according to the home) when the home starts servicing the request, and the *D'-state*, which is the requesting node's new state in the home. The symbol "-" indicates that no state change is necessary and the symbol "*" corresponds to all possible states. If the requester's state is not known, due to a limited directory representation, the D-state is here assumed to be I. State $M_{ODIFIED}$ (M) is when the node is the owner and no sharers exists.

Demand specifies what demand transactions are sent from the home to the slaves. We distinguish between transactions to an owner and to a sharer. H_INV transactions are not sent to the requesting node, but all other transactions are sent if the home node is also a slave. Each demand carries the value of the number of demands sent out by the home agent.

Reply specifies what reply transactions are received by the requester to the home. We distinguish between transactions from an owner, a sharer and from the home. Each reply carries the value of the number of demands sent out by the home agent.

FT Reissue specifies what local interconnect transactions to send after all replies have been received. The extensions to the transactions are explained in Table 11-1. This column also defines the new MTAG state, which is sent with the transaction. The symbol "-" indicates that no state change is performed. Note that this is the only way of changing the MTAG state. This is why sometimes "dummy" RS_N/*new_state* are used to change the MTAG state. Note also that reissued transactions use the "normal" GA or LPA space even if the original transaction was to an alternative space. e.g., WS_PM.

Compl. describes the completion phase. It always involves a packet sent from the request agent back to the home agent. The completion may carry data.

Referring to both the table of Figure 3 and the flowchart of Figure 2A, one of several well-known algorithms can be used to select a suitable page for replacement (step 210). For example, the selection criteria may be a shadow page that has been least-recently used (LRU) or least-frequently used. Note that home pages of subsystem 110 are normally preserved, i.e., they are treated preferentially with respect to shadow pages, since home pages are typically poor candidates for replacement.

Upon selection of a suitable shadow page for replacement, the selected page is demapped locally. In other words, local access to the selected page by processors 111a, 111b, ... 111i is frozen while the selected shadow page is in the process of being "flushed" (step 220). Flushing restores coherency between shadow pages and home pages within system 100 whenever a shadow page is discarded.

In this implementation, since the higher-level cache, e.g., cache 114, maintains MTAGs (memory tags) reflecting the status of the shadow cache lines locally, e.g., in directory 116, these local MTAG entries associated with each shadow cache line of the selected shadow page can be scanned by processor 111a. If the status of one or more of the shadow cache lines is valid, e.g., having an "O" (owned) or an "M" (modified) state, these shadow cache line(s) are identified for flushing (step 230). (See row #1 of Figure 3). Alternatively, the entire selected page, i.e., every cache line associated with the selected page, can be flushed without consulting the local MTAGs, regardless of the respective MTAG state.

Figure 2B details sap 240 for flushing the selected shadow page from cache memory 114 of requesting subsystem 110. The asynchronous flush operation of each valid shadow cache line is carried out in a two distinct asynchronous phases. For each valid cache line in the selected shadow page, a local flush instruction (WS_FLU) which includes the address identifying the shadow cache line is sent to local global interface 115 of requesting subsystem 110 (step 241). In response to each local flush instruction, global interface 115 spawns a global flush instruction (R_FLU) to home subsystem 120 (step 242).

Figure 1B shows an exemplary scheme for encoding flush instructions which use different ranges of addresses so that the flush instructions can be easily distinguished from other instructions by local interconnect 119.

In accordance with the embodiment, by decoupling the local flush instructions from the global flush instructions, once processor 111a is done issuing a local flush instruction, processor 111a does not have to wait for a corresponding response from home subsystem 120. Instead, processor 111a is freed up quickly since there is no need to wait for an acknowledgment from home subsystem 120 over global interconnect 190.

Referring again to Figure 2B, upon receipt of the global flush instruction from requesting subsystem 110, home subsystem 120 sends an appropriate "acknowledgment" message back to requesting subsystem (step 243). As discussed above and detailed in Figure 3, the type of acknowledgment message depends on the status of the shadow cache line as recorded in home directory 126.

As shown in row #1 of Figure 3, if the status of the shadow cache line is "M" or "O", home subsystem 120 sends a "H_ACK" acknowledge message to requesting subsystem 110 indicating that the content of the corresponding cache line in the home page needs to be updated, i.e., the cache line is "dirty". Since requesting subsystem 110 has "dirty" data, home subsystem 120 has to "pull" the dirty data value from the replacing (previously requesting) subsystem 110. Sending the

"H_ACK" causes requesting subsystem 110 to "reissue" a read-to-own (RTO) transaction on its local interconnect 119 (step 244). Because "dirty data can reside in either cache 114, L2$ 112a, or in both caches, the TRO transaction causes the retrieval of the dirty data from the appropriate cache within subsystem 110. The issuance of the RTO transaction on local interconnect 119 also has the effect of invalidating any shared copy within subsystem 110 and also updates the respective local MTAGs to the "I" state.

As shown in row #1 of Figure 3, having retrieved the dirty data via local interconnect 119, requesting subsystem 110 can now send the dirty data appended to a "completion message", e.g., a "R_CMP_W" completion message (step 245). Home subsystem 120 is now able to update its home copy of the data and also its home directory 126 by marking the corresponding entry in home directory 126 "I" (invalid) (step 246). Hence, the above described flush operation permits the "M" and "O" copies of the home page to migrate back to COMA cache memory 124 of home system 120 gracefully and efficiently.

Referring now to row #2 of Figure 3, if the status of the shadow cache line is "S" (shared), home subsystem 120 sends a "H_NACK" acknowledge message to requesting subsystem 110 indicating that the shadow cache line in the shadow page can be discarded and the corresponding MTAG associated with requesting directory 116 can be marked "I". Accordingly, requesting subsystem 110 reissues an RTO transaction on local interconnect 119 thereby invalidating any shared copy within subsystem 110 and also updates the respective local MTAGs of subsystem 110 to the "I" state. A "R_CMP" completion message, without any appended data, is sent to home subsystem 120. Home subsystem 120 updates the corresponding entry in home directory 126 by marking the shadow cache line as having an "I" (invalid) state.

Conversely, as depicted in row #3 of Figure 3, if the status of the shadow cache line in home directory 126 is "I", home subsystem 120 sends a "H_NOPE" message to requesting subsystem 110. Subsequently, requesting subsystem 110 discards the shadow cache line and marks its corresponding local MTAG "I". A "R_CMP" completion message (without data) is sent to home subsystem 120.

As shown in row #4 of Figure 3, if the local MTAG of requesting subsystem 110 shows the copy of data to be "I", no further action is taken by both requesting subsystem 110 and home subsystem 120.

As discussed above, it is also possible for data to be cached in L2$ 113a but not in cache 114, for example, if data is cached solely in NUMA mode. Hence, in addition to gracefully migrating COMA copies (using a local physical address) to home subsystem 120, system 100 has also has to be capable of gracefully migrating NUMA copies (using a global address) back to home subsystem 110. This can be accomplished by generating the appropriate NUMA and/or COMA flush instructions using the global and local flush address space (encoding) shown in Figure 1B.

Referring to rows #5-7 of Figure 3, since the issuance of an RTO transaction on local interconnect 119 causes all copies of the data to be retrieved, the steps taken by requesting subsystem 110 is similar to the above-described steps for flushing data stored in cache 114. In this implementation, the major difference being the MTAGs only record the status of data stored in cache 114 but does not reflect the status of data stored in L2$ 113a. Hence, when data stored in L2$ 112a is replaced, with the exception of updating the local MTAG of requesting subsystem 110, both requesting subsystem 110 and home subsystem 120 take similar actions described above and depicted in row #1-3 of Figure 3.

Referring again to step 250 of Figure 2A, a confirmation of the completion of the flush operation can be implemented using a "synchronization" mechanism. One such confirmation verifies that all previously valid cache lines associated with a page have been successfully replaced with respect to their home location and the replaced cache lines are now marked "invalid". An exemplary synchronization mechanism is described below.

Turning next to Figs. 4 and 5, a block diagram of one embodiment of global interface 115 and a detailed block diagram of request agent 400 are shown, respectively. Additionally, SMP in queue 94, SMP PIQ 96, SMP out queue 92, and transaction filter 98 are shown. Transaction filter 98 is coupled to SMP bus 20, SMP in queue 94, SMP PIQ 96, and request agent 400. SMP out queue 92, SMP in queue 94, and SMP PIQ 96 are coupled to request agent 400 as well.

Each transaction presented upon SMP bus 20 for which ignore signal 70 is asserted is stored by global interface 115 for later reissue. As mentioned above, ignore signal 70 may be asserted if the access rights for the affected coherency unit do not allow the transaction to complete locally. Additionally, ignore signal 70 may be asserted if a prior transaction from the same subnode 50 is pending within global interface 115. Still further, ignore signal 70 may be asserted for other reasons (such as full in queues, etc.).

Request agent 400 comprises multiple independent control units 310A-310N. A control unit 310A-310N may initiate coherency activity (e.g. perform a coherency request) for a particular transaction from SMP in queue 94 or SMP PIQ 96, and may determine when the coherency activity completes via receiving replies. An initiation control unit 312 selects transactions from SMP in queue 94 and SMP PIQ 96 for service by a control unit 310A-310N. Any selection criteria may be employed as long as neither SMP in queue 94 nor SMP PIQ 96 are unconditionally prioritized higher than the other and as long as at least one control unit 310A-310N is not allocated to performing I/O operations.

In addition to selecting transactions for service by

control units 310, initiation control unit 312 informs a second control unit 314 that a synchronization operation has been selected for initiation. A sync signal upon a sync line 316 coupled between initiation control unit 312 and control unit 314 is asserted when a synchronization operation is selected from either SMP in queue 94 or SMP PIQ 96. Control unit 314 manages a synchronization vector control register 318, and reissues the synchronization operation to SMP out queue 92 upon completion of the synchronization operation.

Upon receipt of an asserted sync signal upon sync line 316, control unit 314 causes control register 318 to record which control units 310 are currently performing coherency activities (i.e. those control units 310 which are not idle). In one embodiment, control register 318 includes multiple bits. Each bit corresponds to one of control units 310. If the bit is set, the corresponding control unit 310A-310N is performing coherency activity which was initiated prior to control unit 314 initiating a synchronization operation. If the bit is clear, the corresponding control unit 310A-310N is either idle or performing coherency activity which was initiated subsequent to control unit 314 initiating a synchronization operation. Each control unit 310 provides an idle line (e.g. idle line 322A from control unit 310A) to control register 318. When the idle signal upon an idle line 322 is asserted, the bit corresponding to the idle control unit 310 within control register 318 is cleared.

Control unit 314 monitors the state of control register 318. When each of the bits have been reset, each of control units 310 have been idle at least once. Therefore, coherency activity which was outstanding upon initiation of the synchronization operation has completed. Particularly, the transactions corresponding to the coherency activity have been globally performed. Therefore, the synchronization operation is complete. Control unit 314 reissues the synchronization operation to SMP out queue 92, and subsequently the reissue transaction completes within the SMP node. More particularly, the synchronization transaction is cleared from the initiating processor. The initiating processor may therefore determine when the synchronization operation has completed (by inserting a processor level synchronization subsequent to the synchronization operation, for example). Exemplary code sequences employing the synchronization operation are shown below.

In one embodiment, the synchronization operation is placed into SMP in queue 94 upon performance of the synchronization operation upon SMP bus 20 (similar to other transactions). Additionally, ignore signal 70 is asserted for the synchronization operation upon SMP bus 20.

It is noted that request agent 400 is configured to accept only one synchronization operation at a time in the present embodiment. Furthermore, two types of synchronization operations are defined: a coherent synchronization and I/O synchronization. Coherent synchronizations synchronize transactions placed in SMP in queue 94. Alternatively, I/O synchronizations synchronize I/O transactions (i.e. transactions placed in SMP PIQ 96).

Additionally, control units 310 may further employ a freeze state for use when errors are detected. If an error is detected for a transaction being serviced by a control unit 310, the control unit transitions to a freeze state and remains therein until released by a software update to a control register. In this manner, information regarding the transaction for which the error is detected (stored by the state machine) may be accessed to aid in determining the error. For purposes of allowing synchronization operations to complete, entering the freeze state is equivalent to entering the idle state.

Turning next to Fig. 6, a table 330 is shown listing exemplary asynchronous operations according to one embodiment of computer system 100. A column 332 lists the asynchronous transaction. A column 334 lists the encoding of the transaction upon SMP bus 20. Finally, a column 336 lists the synchronization operation which is used to synchronize the particular asynchronous operations.

The fast write stream asynchronous operation is employed to enhance the performance characteristics of writes to remote nodes. When a fast write stream operation is performed system interface 115 allows the initiating processor to transfer the data thereto prior to performing coherency activities which may be required to obtain write permission to the affected coherency unit. In this manner, the processor resources consumed by the fast write stream operation may be freed more rapidly than otherwise achievable. As shown in column 334, the fast write stream operation is coded as a write stream having the five most significant bits of the address coded as shown. The "nn" identifies the home node of the address. The coherent synchronization operation ("WS_SC") is used to synchronize the fast write stream operation.

A second asynchronous operation employed in the exemplary embodiment is the flush operation. When a flush operation is detected by system interface 115, the affected coherency unit (if stored in the SMP node) is flushed. In other words, the coherency unit is stored back to the home node and the MTAG for the coherency unit is set to invalid. In the exemplary embodiment, the flush operation is coded as a write stream operation having the five most significant bits of the address coded as shown in column 334. The flush command uses a write stream encoding, although the data corresponding to the write stream is discarded. Similar to the fast write stream, system interface 115 allows the data to be transferred prior to global performance of the flush operation. The flush operation is synchronized using WS_SC.

The synchronization operations in the exemplary embodiment are coded as write stream operations as well, although any encoding which conveys the syn-

chronization command upon SMP bus 20 may be used. In particular for the exemplary embodiment, the WS_SC operation is coded as a write stream operation for which the seven most significant address bits are coded as 0111100 (in binary). The WS_SP operation is coded as a write stream operation for which the seven most significant address bits are coded as 0111101 (in binary). An alternative embodiment may employ a specially coded I/O read operation to perform synchronization. When the I/O read operation is detected, previously received transactions are completed prior to returning data for the I/O read operation.

Turning now to Fig. 7, an exemplary code sequence 340 is shown depicting use of synchronization operations. The example includes instructions from the SPARC microprocessor architecture. The order of operations in the program (the "program order") is indicated by arrow 342. In the example, several fast write stream operations are performed (the "WS_FW" operations shown in Fig. 7). Upon completion of a series of fast write stream operations, the code sequence includes a WS_SC operation to synchronize the completion of the operations. Additionally, a MEMBAR instruction is Included to guarantee completion of the WS_SC operation prior to initiation of any memory operations subsequent to the MEMBAR instruction.

Generally, the WS_SC operation is an example of a system level synchronization operation. The WS_SC operation causes a synchronization to occur in the system Interface 115 of the SMP node 12A-12D within which the WS_SC operation is executed. In this manner, the node is synchronized. However, synchronizing the processor itself is performed using a processor level synchronization operation. The processor level synchronization operation does not synchronize the node, but does synchronize the processor 111a within which it is executed. By pairing a system level synchronization in the manner of Fig. 7, a complete synchronization of each level of the computer system may be achieved.

Various optimizations or improvements of the above described cache coherent mechanism are possible. For example, when flushing a shadow page, instead of flushing valid cache lines individually, the entire page may be flushed. Performance tradeoffs are also possible. For example, instead of flushing cache lines with a "M" or "O" state when a page is replaced, the entire page, i.e., every cache line may be flushed, simplifying the procedure at the expense of the network traffic.

Other modifications and additions are possible without departing from the invention. For example, instead of blocking all read and write requests whenever a request is outstanding, read-to-share requests are blocked only if there is a read-to-own or a write-back request outstanding. In addition, each subsystem may be equipped with additional circuitry to perform "local data forwarding" so that processors within a subsystem can provide data to each other without accessing the host directory of another subsystem. Hence, the scope of the invention should be determined by the following claims.

Particular and preferred aspects of the invention are set out in the accompanying independent and dependent claims. Features of the dependent claims may be combined with those of the independent claims as appropriate and in combinations other than those explicitly set out in the claims.

Claims

1. A method for replacing data while maintaining coherency of said data within a computer system having at least a first and second subsystem coupled to each other via an interconnect, wherein said first subsystem is a home subsystem of said data, each said subsystem including a cache and a global interface, and each said cache include a plurality of cache lines, the method comprising the steps of:

   caching a copy said data in an attraction cache line of said cache of said second subsystem, said second subsystem thereby becoming a requesting subsystem of said data;
   detecting a need to replace said data in said attraction cache line of said requesting subsystem; and
   asynchronously flushing said copy of said data from said attraction cache line of said requesting subsystem while maintaining coherency of said data within said computer system.

2. The method of claim 1 wherein said asynchronous flushing step comprises the steps of:

   locally demapping said attraction cache line of said requesting subsystem;
   issuing a local flush instruction to the global interface of said requesting subsystem;
   issuing a global flush instruction from said global interface of said requesting subsystem to the global interface of said home subsystem in response to said local flush instruction; and
   determining that the coherency of said data has been maintained between said home subsystem and said requesting subsystem.

3. The method of claim 1 wherein said coherency determining step comprises the steps of:

   sending a synchronization request from said requesting subsystem to said home subsystem;
   verifying that said data is now coherent between said home subsystem and said requesting subsystem; and
   sending an acknowledgment from said home

subsystem to said requesting subsystem in response to said synchronization request, said acknowledgment indicating that said data is now coherent between said requesting and said home subsystem.

4. The method of claim 2 wherein said data has a modified or an owned state, said method further comprising the step of sending an acknowledgment from said home subsystem in response to said global flush instruction, said acknowledgment indicating that said copy of said data in said requesting subsystem may have been modified and should be sent to said home subsystem.

5. The method of claim 2 wherein said data has a shared or an invalid state, said method further comprising the step of sending an acknowledgment from said home subsystem in response to said global flush instruction, said acknowledgment indicating that said copy of said data in said requesting subsystem can be discarded.

6. A mechanism for replacing data while maintaining coherency of said data within a computer system having at least a first and second subsystem coupled to each other via an interconnect, wherein said first subsystem is a home subsystem of said data, and said data is cached in an attraction cache line of said cache of said second subsystem, said second subsystem thereby becoming a requesting subsystem of said data, and wherein each said subsystem including a cache and a global interface, and each said cache include a plurality of cache lines, the mechanism comprising:

a detector configured to detect a need to replace said data in said attraction cache line of said requesting subsystem; and
an asynchronous flusher configured to replace said data from said attraction cache line of said requesting subsystem while maintaining coherency of said data within said computer system.

7. The mechanism of claim 6 wherein said asynchronous flusher is further configured to locally demap said attraction cache line and configured to issue a local flush instruction to the global interface of said requesting subsystem, said global interface of said requesting subsystem is configured to issue a global flush instruction to the global interface of said home subsystem in response to said local flush Instruction, and said global interface of said home subsystem is configured to determine that the coherency of said data has been maintained between said home subsystem and said requesting subsystem.

8. The mechanism of claim 7 wherein said asynchronous flusher is further configured to send a synchronization request from said requesting subsystem to said home subsystem, configured to verify that said data is now coherent between said home subsystem and said requesting subsystem, and configured to send an acknowledgment from said home subsystem to said requesting subsystem in response to said synchronization request.

9. Computer apparatus comprising a mechanism according to any one of claims 6 to 8.

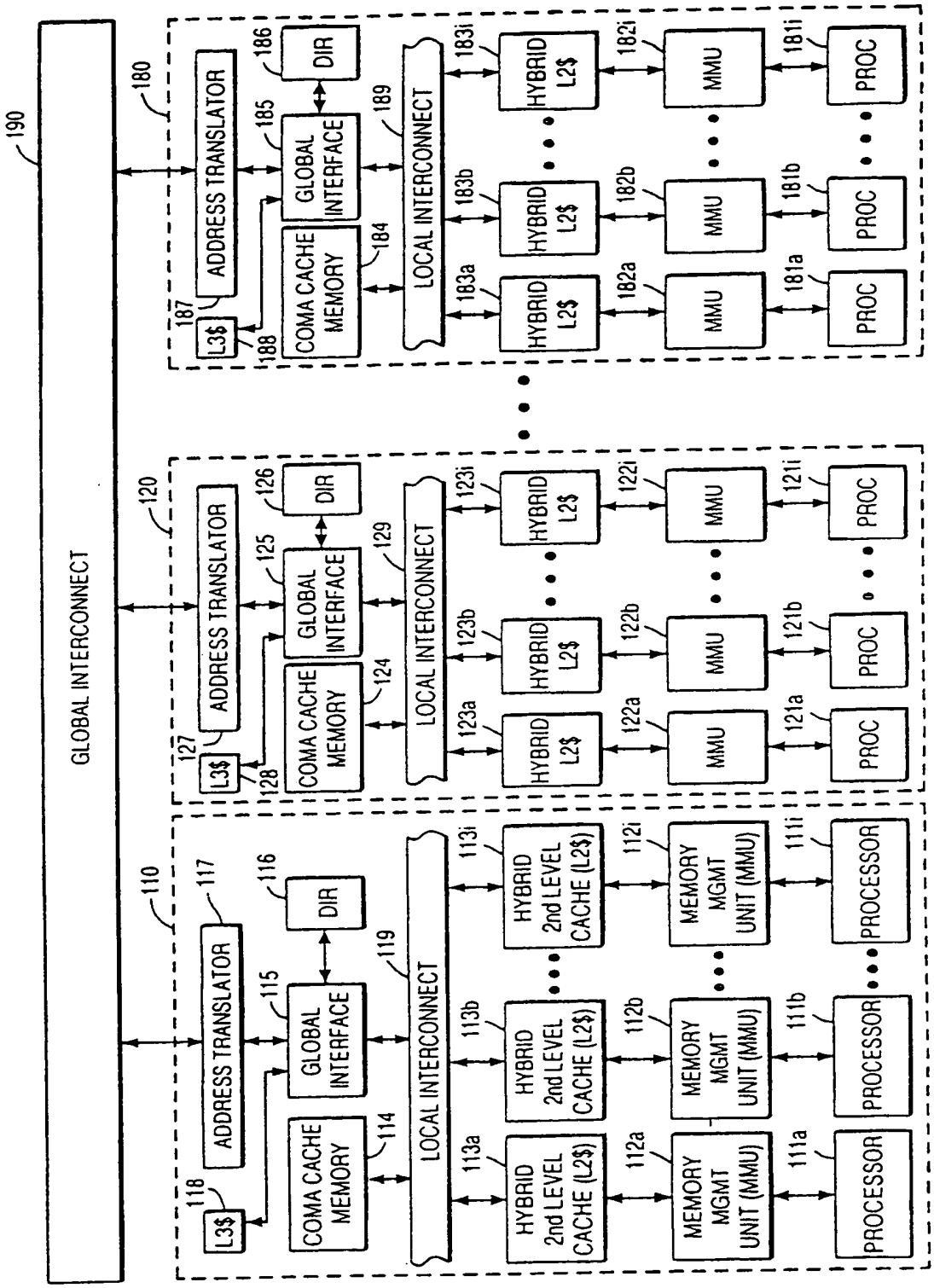
GLOBAL INTERCONNECT
190
100
110
120
180
L3$
118
ADDRESS TRANSLATOR
117
COMA CACHE MEMORY
114
GLOBAL INTERFACE
115
DIR
116
LOCAL INTERCONNECT
119
HYBRID 2nd LEVEL CACHE (L2$)
113a
113b
113i
MEMORY MGMT UNIT (MMU)
112a
112b
112i
PROCESSOR
111a
111b
111i
127
128
124
125
126
129
123a
123b
123i
HYBRID L2$
122a
122b
122i
MMU
121a
121b
121i
PROC
187
188
184
185
186
189
183a
183b
183i
182a
182b
182i
181a
181b
181i

FIG. 1A

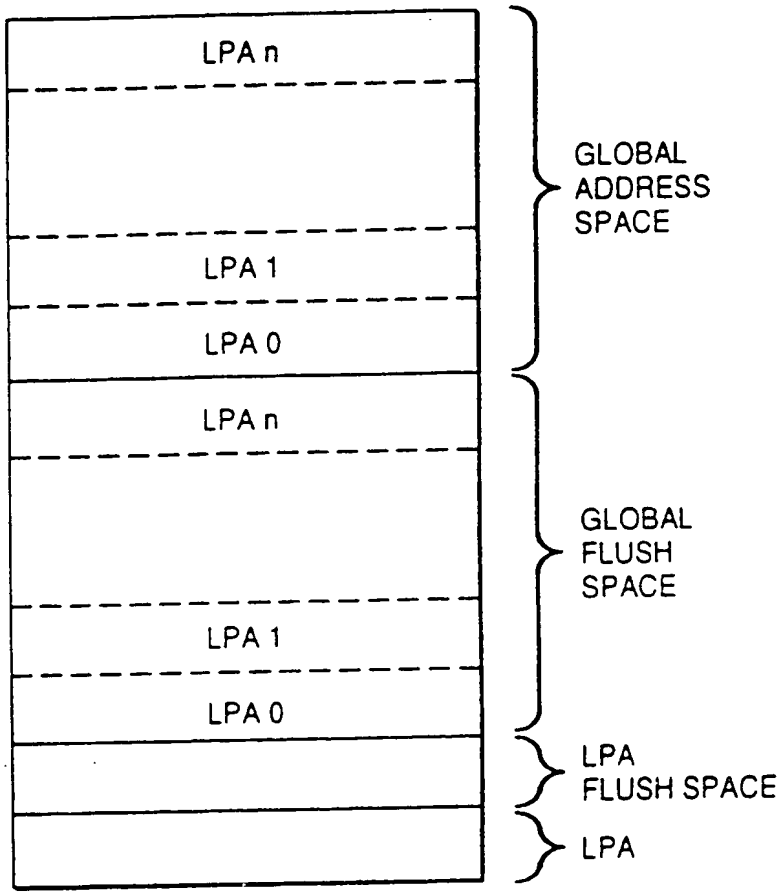
LPA n
LPA 1
LPA 0
GLOBAL ADDRESS SPACE
LPA n
LPA 1
LPA 0
GLOBAL FLUSH SPACE
LPA FLUSH SPACE
LPA

FIG. 1B

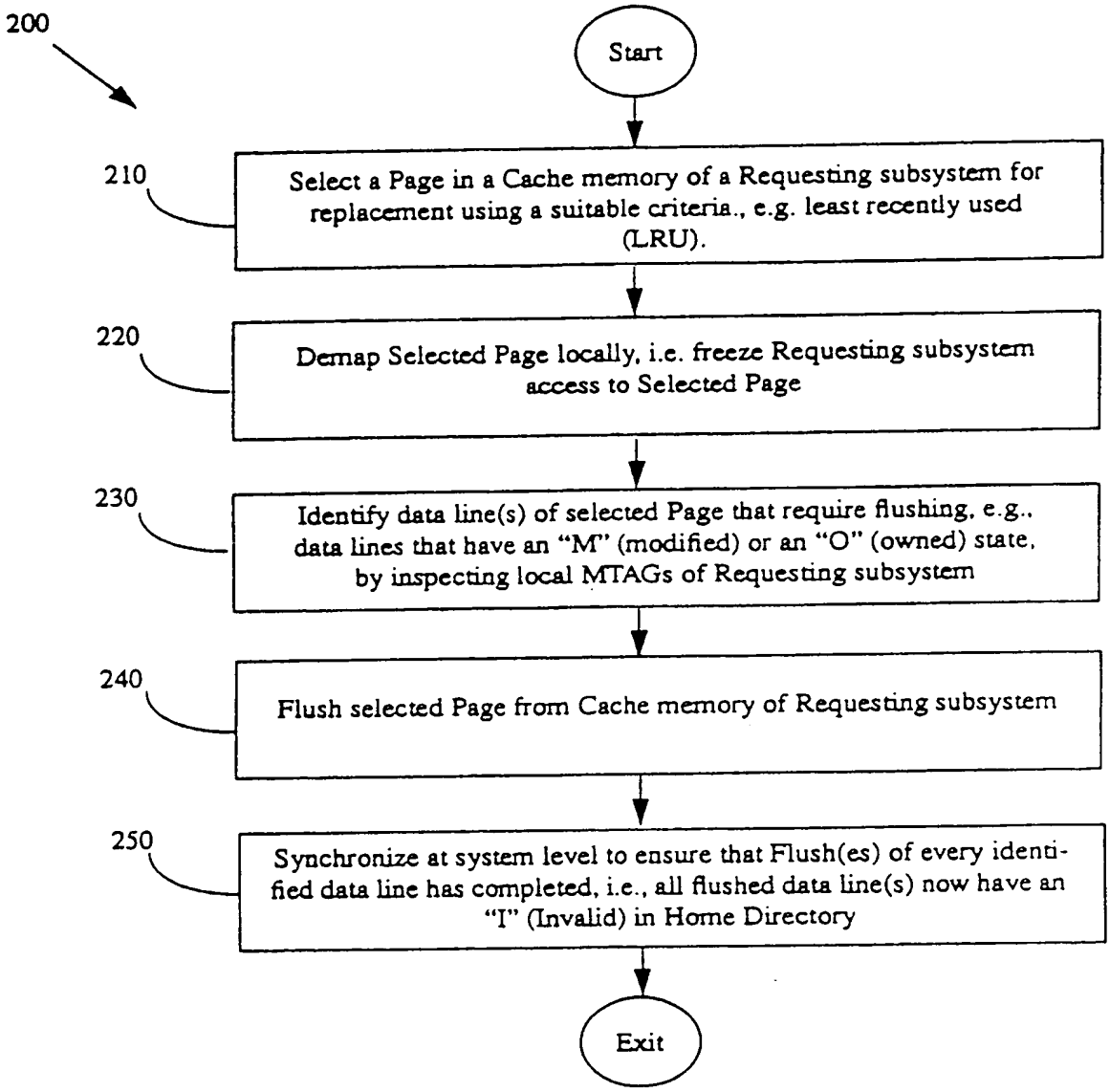
200
Start
210
Select a Page in a Cache memory of a Requesting subsystem for replacement using a suitable criteria., e.g. least recently used (LRU).
220
Demap Selected Page locally, i.e. freeze Requesting subsystem access to Selected Page
230
Identify data line(s) of selected Page that require flushing, e.g., data lines that have an "M" (modified) or an "O" (owned) state, by inspecting local MTAGs of Requesting subsystem
240
Flush selected Page from Cache memory of Requesting subsystem
250
Synchronize at system level to ensure that Flush(es) of every identified data line has completed, i.e., all flushed data line(s) now have an "I" (Invalid) in Home Directory
Exit

FIG. 2A

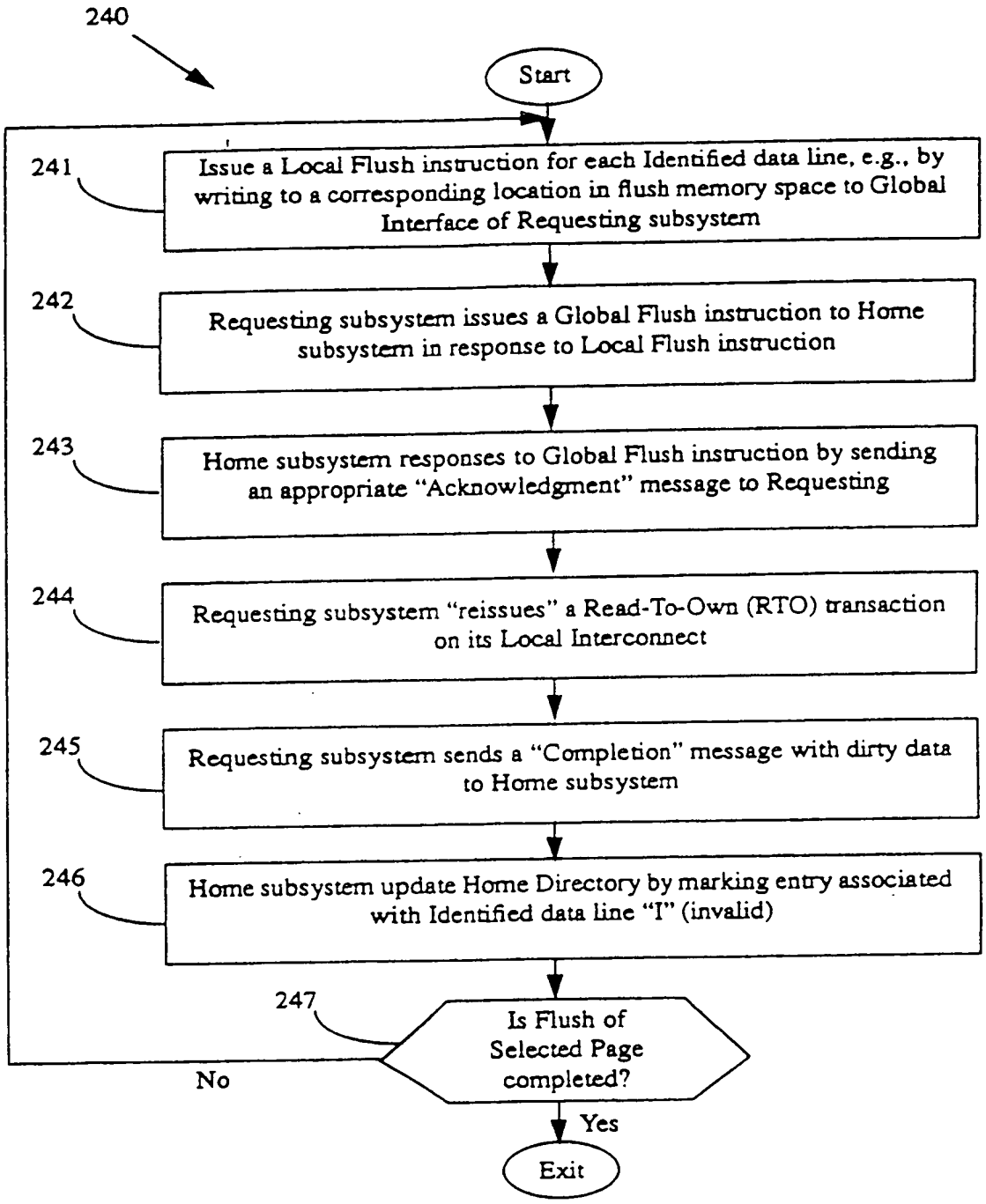
240
Start
241
Issue a Local Flush instruction for each Identified data line, e.g., by writing to a corresponding location in flush memory space to Global Interface of Requesting subsystem
242
Requesting subsystem issues a Global Flush instruction to Home subsystem in response to Local Flush instruction
243
Home subsystem responses to Global Flush instruction by sending an appropriate "Acknowledgment" message to Requesting
244
Requesting subsystem "reissues" a Read-To-Own (RTO) transaction on its Local Interconnect
245
Requesting subsystem sends a "Completion" message with dirty data to Home subsystem
246
Home subsystem update Home Directory by marking entry associated with Identified data line "I" (invalid)
247
Is Flush of Selected Page completed?
No
Yes
Exit

FIG. 2B

ROW	Request Agent Protocol			Home Agent Protocol					Slave A. P.		Req. Agent Protocol	
	Bus Trans	Req. Node MTAG	Request: From Req. To Home	State in Dir: D	State in Dir: D'	Demand: From: Home: To Owner	Demand: From: Home: To Sharer	Reply: To Req.	Owner: To Req.	Sharer: To Req.	FT Reissue: Trans/ MTAG'	Compl.: From Req. To Dir
#1	WS_FLU	M,O, S	R_FLU	M,O	I	-	-	H_ACK	-	-	RTO_N_E/I	R_CMP_W
#2				S	I	-	-	H_NACK	-	-	RTO_N_S/I	R_CMP
#3				I	-	-	-	H_NOPE	-	-	-	R_CMP
#4	- - - - -	I	-	-	-	-	-	-	-	-	-	-
#5		N	R_FLU	M,O	I	-	-	H_ACK	-	-	RTO_N_E/-	R_CMP_W
#6				S	I	-	-	H_NACK	-	-	RTO_N_S/-	R_CMP
#7				I	-	-	-	H_NOPE	-	-	-	R_CMP

FIG. 3

EP 0 817 081 A2

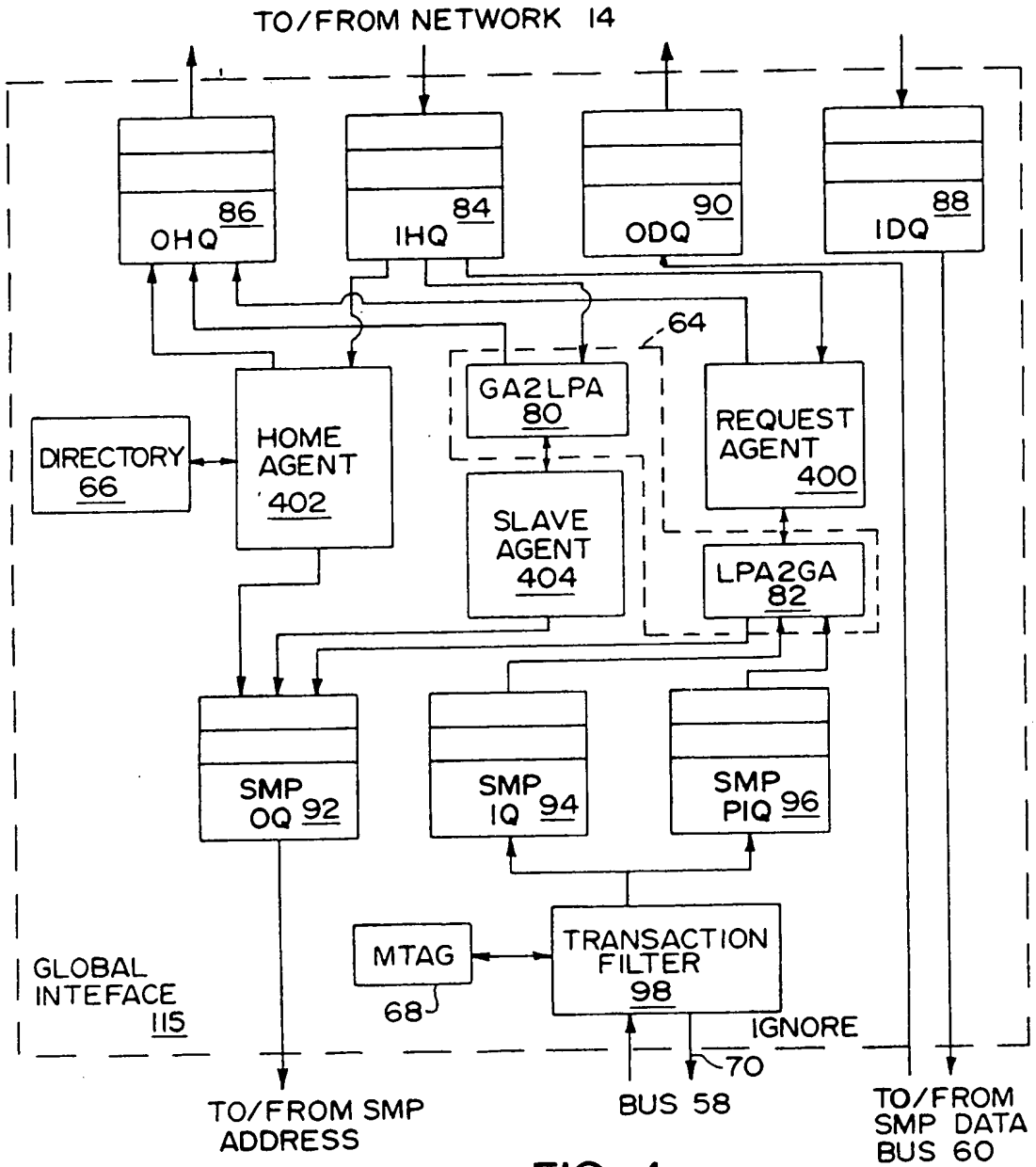
TO/FROM NETWORK 14
OHQ 86
IHQ 84
ODQ 90
IDQ 88
64
GA2LPA 80
REQUEST AGENT 400
DIRECTORY 66
HOME AGENT 402
SLAVE AGENT 404
LPA2GA 82
SMP OQ 92
SMP IQ 94
SMP PIQ 96
MTAG
68
TRANSACTION FILTER 98
IGNORE
70
GLOBAL INTEFACE 115
TO/FROM SMP ADDRESS
BUS 58
TO/FROM SMP DATA BUS 60

FIG. 4

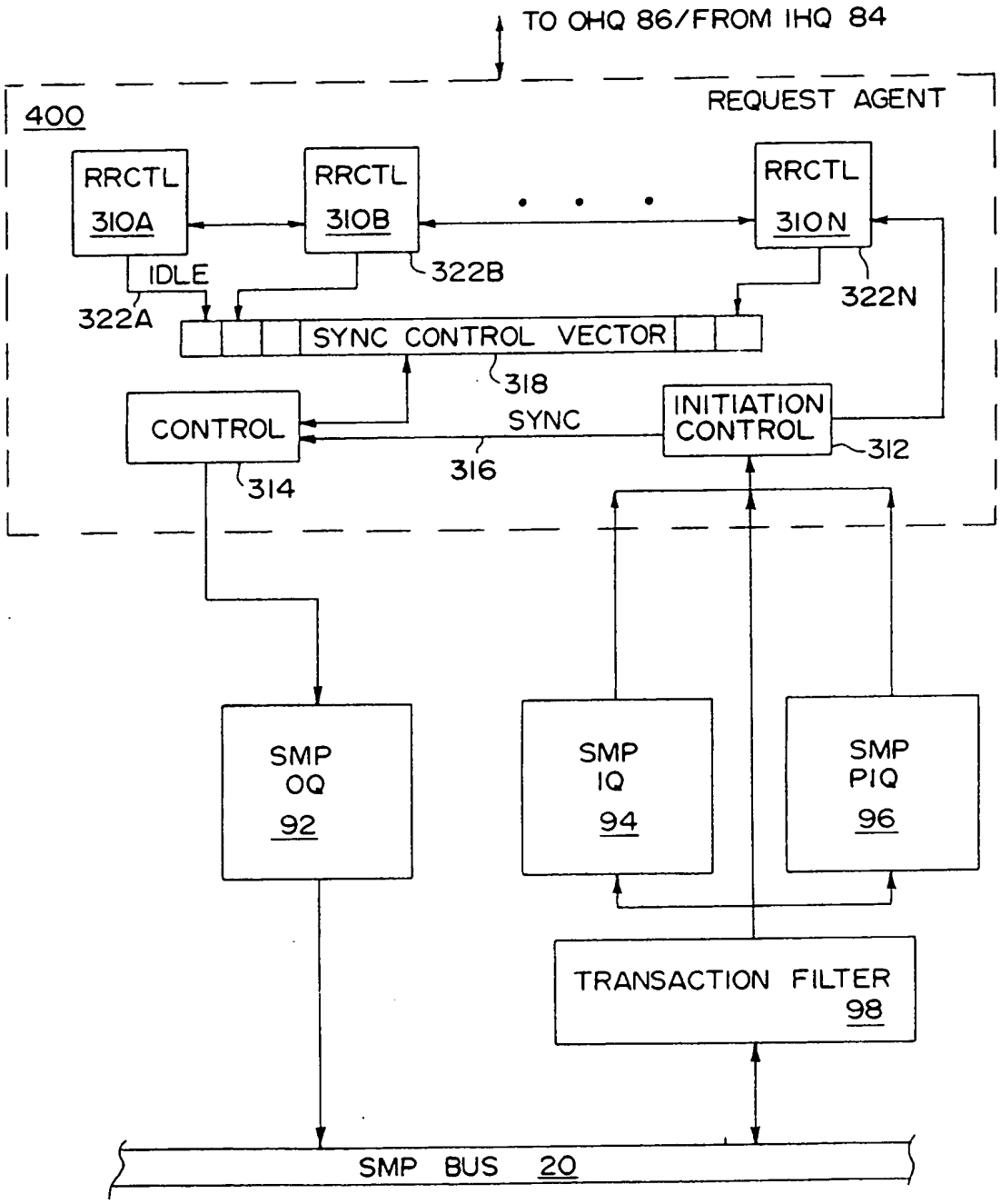
TO OHQ 86/FROM IHQ 84
400
REQUEST AGENT
RRCTL 310A
RRCTL 310B
RRCTL 310N
IDLE
322A
322B
322N
SYNC CONTROL VECTOR
318
CONTROL
314
SYNC
316
INITIATION CONTROL
312
SMP OQ 92
SMP IQ 94
SMP PIQ 96
TRANSACTION FILTER 98
SMP BUS 20

FIG. 5

Transaction (332)	Encoding (334)	Synchronization (336)
Fast Write Stream	WS, LPA=0nn10...	WS_SC
Flush	WS, LPA=0nn01...	WS_SC
Prefetch	WS, LPA=00m11...	WS_SC
Copy	WS, LPA=01011...	WS_SC
Administrative	ADM	WS_SC
I/O writes	WIO, WBIO	WS_SP

330

FIG. 6

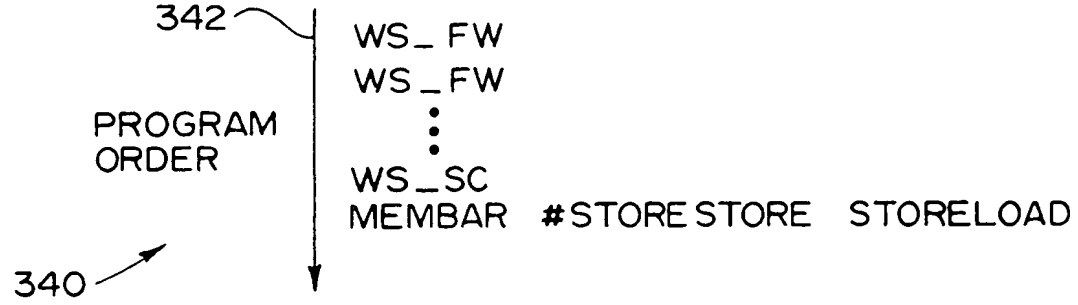

FIG. 7

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 817 081 A3

(12) EUROPEAN PATENT APPLICATION

(88) Date of publication A3:
04.02.1998 Bulletin 1998/06

(43) Date of publication A2:
07.01.1998 Bulletin 1998/02

(21) Application number: **97304796.2**

(22) Date of filing: **30.06.1997**

(51) Int. Cl.[6]: **G06F 12/08**

(84) Designated Contracting States:
AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE

(30) Priority: **01.07.1996 US 673881**

(71) Applicant:
SUN MICROSYSTEMS, INC.
Mountain View, California 94043-1100 (US)

(72) Inventors:
- **Hagersten, Erik E.**
  Palo Alto, California 94303 (US)
- **Guzovskiy, Aleksandr**
  Lowell, Massachusetts 01852 (US)

(74) Representative:
Turner, James Arthur et al
D. Young & Co.,
21 New Fetter Lane
London EC4A 1DA (GB)

(54) Flushing of cache memory in a computer system

(57) An efficient streamlined cache coherent protocol for replacing data is provided in a multiprocessor distributed-memory computer system. In one implementation, the computer system includes a plurality of subsystems, each subsystem includes at least one processor and an associated cache and directory. The subsystems are coupled to a global interconnect via global interfaces. In one embodiment, when data is replaced from a requesting subsystem, an asynchronous flush operation is initiated. In this implementation, the flush operation includes a pair of decoupled local flush instruction and corresponding global flush instruction. By decoupling the local flush instructions from the global flush instructions, once the requesting processor in the requesting subsystem is done issuing the local flush instruction, the requesting processor does not have to wait for a corresponding response from home location associated with the data being replaced. Instead, the requesting processor is freed up quickly since there is no need to wait for an acknowledgment from the home location (home subsystem) over the global interconnect. The home subsystem responds with an appropriate ACK message. The requesting subsystem reissues a read-to-own (RTO) transaction on its local interconnect thereby retrieving and invalidating any copy(s) of the data in the requesting subsystem. A Completion message is sent to the home subsystem together with the dirty data. Subsequently, a confirmation of the completion of the flush operation can be implemented using a "synchronization" mechanism to verify that all previously valid cache lines associated with a page have been successfully replaced with respect to their home location and the replaced cache lines are now marked "invalid" at the home subsystem.

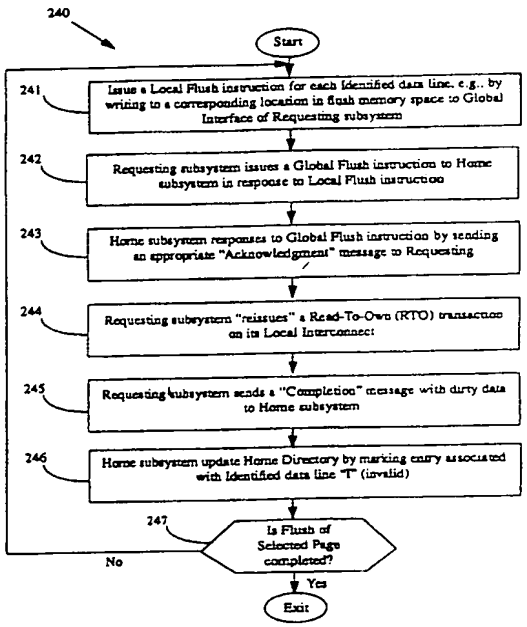

FIG. 2B

EP 0 817 081 A3

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 97 30 4796

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
A	SAULSBURY A ET AL: "AN ARGUMENT FOR SIMPLE COMA" FUTURE GENERATIONS COMPUTER SYSTEMS, vol. 11, no. 6, 1 October 1995, pages 553-566, XP000536147 * page 559, right-hand column, line 36 - page 560, right-hand column, line 42 *	1-9	G06F12/08
A	"FORCE PURGE TABLE FOR MEMORY DIRECTORY IN DIRECTORY BASED SYMMETRIC MULTIPROCESSOR SYSTEMS" IBM TECHNICAL DISCLOSURE BULLETIN, vol. 37, no. 8, 1 August 1994, page 293/294 XP000456422 * the whole document *	1-9	
A	JOE T ET AL: "EVALUATING THE MEMORY OVERHEAD REQUIRED FOR COMA ARCHITECTURES" COMPUTER ARCHITECTURE NEWS, vol. 22, no. 2, 1 April 1994, pages 82-93, XP000450341 * page 83, right-hand column, line 11 - page 84, right-hand column, line 2 *	1,6	TECHNICAL FIELDS SEARCHED (Int.Cl.6) G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	9 December 1997	Nielsen, O

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)